

Fall 10-2015

Porting the NetBeans Java 8 Enhanced For Loop Lambda Expression Refactoring to Eclipse

Md Arefin

CUNY New York City College of Technology

Raffi T. Khatchadourian

CUNY Hunter College

How does access to this work benefit you? Let us know!

Follow this and additional works at: https://academicworks.cuny.edu/hc_pubs

 Part of the [Programming Languages and Compilers Commons](#), and the [Software Engineering Commons](#)

Recommended Citation

Md. Arefin and Raffi Khatchadourian. Porting the netbeans Java 8 enhanced for loop lambda expression refactoring to Eclipse. In Companion Proceedings of the 2015 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity, SPLASH Companion 2015, pages 58–59, New York, NY, USA, October 2015. ACM.

This Poster is brought to you for free and open access by the Hunter College at CUNY Academic Works. It has been accepted for inclusion in Publications and Research by an authorized administrator of CUNY Academic Works. For more information, please contact AcademicWorks@cuny.edu.



Porting the NetBeans Java 8 Enhanced For Loop Lambda Expression Refactoring to Eclipse



Md Arefin

Mentor: Raffi Khatchadourian

New York City College of Technology, City University of New York



Abstract:

Java 8 is one of the most significant upgrades to Java programming language and framework. This upgrade provided supports for functional programming, a new JavaScript engine, a new APIs for date time manipulation, a new streaming API and so on. There are several new key features of Java 8 that can help make programs easier to read, write, and maintain. Among the new Java 8 features, lambda expressions are touted to be most significant. Lambda expression simplify the development process by facilitating functional programming. Eclipse is one of the most popular IDEs for Java. While JDT UI has incorporated several Java 8 quick-fixes and refactoring's, there are still many features left to be done. For example, NetBeans has a refactoring that converts loops to Lambda expressions. This project is for exploring the porting of such conversion mechanisms in NetBeans to Eclipse. These may be manifested as refactoring and/or quick-fixes. In this project we are working on the Convert to a lambda expression Java 8 refactoring/quick-fixes. (<http://openlab.citytech.cuny.edu/groups/port-netbeans-lambda-refactoring-to-eclipse>)

Examples

```
List<String> list = Arrays.asList("Moe", "Larry", "Curly");

//Regular Enhance For Loop Conversion:
for (String s : list) System.out.println(s);

//Suggested Java 8 Lambda expression:
list.forEach(s -> System.out.println(s));

//Better yet:
list.forEach(System.out::println);

//It also should work with streams:
list.stream.forEach(System.out::println);
```

Eclipse Refactoring Model:

- The API for refactoring provides a process-level abstraction upon which specific refactorings may be built.
- Figure 1 shows the elements of this abstraction at a very high level. Here, arrows between elements represent dependencies

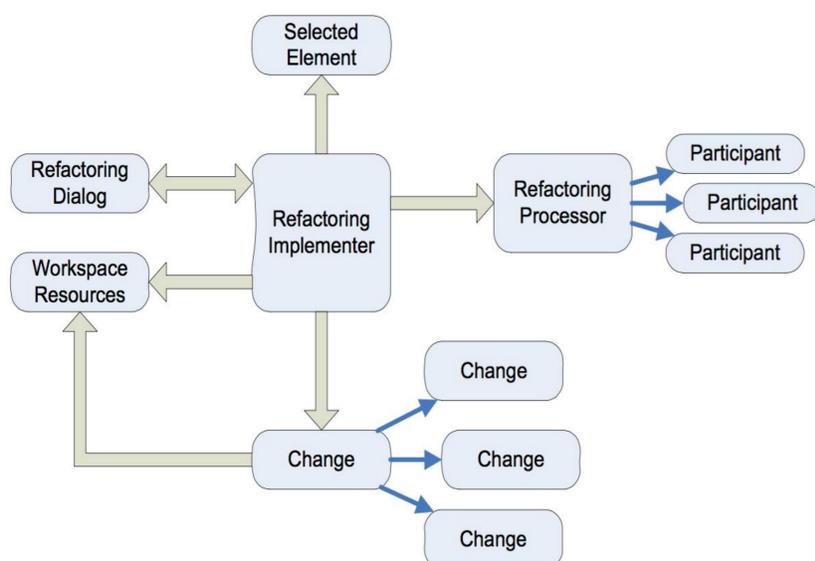


Figure 1: Refactoring API Elements[2]

IMPLICATION & PRECONDITIONS:

- The conversion must be semantics-preserving. That is the behavior of the program prior to the refactoring must match that of after the refactoring.
- The **for** loop must iterate over an instance of an Collection.
- The body of the initial **for** loop must not throw a checked exception.
- The body of the initial **for** loop must not have more than one reference to a local, non-effectively final variable defined outside the loop.
- The **for** loop body must not contain a break or continue statement as these semantics cannot be expressed via a lambda expression.

ACKNOWLEDGMENT:

- We would like to thank Google Summer of Code, as well as the Eclipse Foundation, for their funding towards this project.

References:

[1] Alex Gyori, Lyle Franklin, Danny Dig "Crossing the Gap from Imperative to Functional Programming through Refactoring" (2013).
[2] Petito, Michael "Eclipse Refactoring", Clarkson University, EE564 Spring 2007.

Motivation

- Lambda expression simplify the development process by facilitating functional programming.
- Lambda expression provide concise way to represent method interface and also make it easier to iterate through, filter, and extract data from Collection.
- Due to Java 8 upgrade some programming platform like tomcat, glassfish, groovy base on java need to be upgraded. According to Gyori et al. [1], making such changes manually would require changing approx. 1,700 line of non-commented, non-blank lines of code across approx. 100 files per project, on average. With our plugin, Eclipse developers would not need to make these changes manually.

Eclipse Refactoring Life-Cycle

- Once a refactoring has been initiated, an implementer of that refactoring is used to coordinate condition checking, gathering details about the refactoring, and ultimately producing a series of changes that may be applied to the Eclipse workspace to accomplish the refactoring.
- This implementer must extend the abstract class *org.eclipse.ltk.core.refactoring.Refactoring*. The life-cycle for this class is shown in Figure 2.

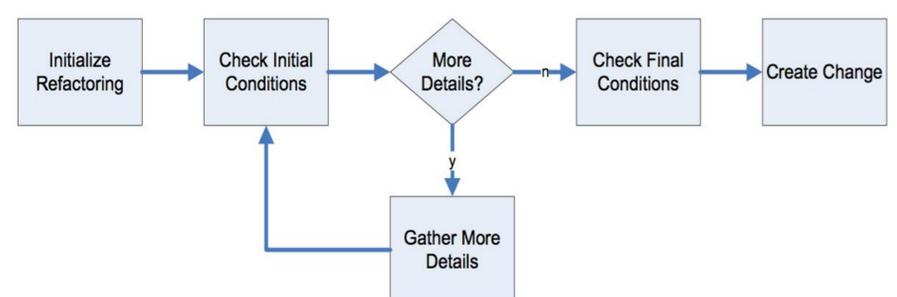


Figure 2: Refactoring Life-Cycle[2]

JDT Type Hierarchy

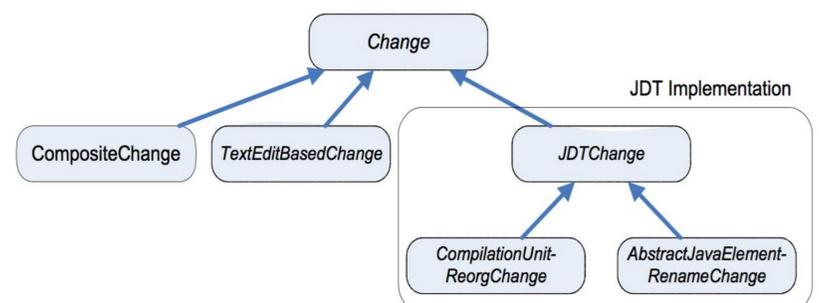


Figure 4: Change Type Hierarchy[2]

FUTURE WORK:

- This is an on-going projects and we will be working on it throughout the fall semester;
- We also plan to develop refactorings for Java 8 that do not currently exist in any IDE.